A Fast and Memory-Efficient Hierarchical Graph Clustering Algorithm*

László Szilágyi[1], Sándor Miklós Szilágyi[2], and Béat Hirsbrunner[3]

[1] Dept. of Control Engineering and Information Technology, Budapest University of Technology and Economics, Hungary
lazacika@yahoo.com
[2] Dept. of Informatics, Petru Maior University of Tîrgu-Mureş, Romania
[3] University of Fribourg, Switzerland

Abstract. In this paper we propose a quick and memory-efficient implementation of the TRIBE-MCL clustering algorithm, suitable for accurate classification of large-scale protein sequence data sets. A symmetric sparse matrix structure is introduced that can efficiently handle most operations of the main loop. The reduction of memory requirements is achieved by regrouping the operations performed during the expansion matrix squaring. The proposed algorithm is tested on synthetic protein sequence data sets of up to 250 thousand items. The validation process revealed that the proposed method performs in 30% less time than previous efficient Markov clustering algorithms, without losing anything from the partition quality. This novel implementation makes it possible for the user of an ordinary PC to process protein sequences sets of 100,000 items in reasonable time.

Keywords: Protein sequence clustering, Markov clustering, Markov processes, efficient computing, sparse matrix.

1 Introduction

Markov clustering performs a hierarchical grouping of input data based on a graph structure and its associated connectivity matrix. When the input data consists of protein sequences, each sequence will be associated to a node of the graph, and edge weights will be the pairwise similarity values computed with existing alignment methods like: Needleman-Wunsch [5], Smith-Waterman [6], and BLAST [1]. In case of large-scale data sets, the BLAST similarity measures are preferred due to its sparse nature, which allows for quick and memory-efficient processing.

TRIBE-MCL is a clustering method based on Markov chain theory [3], which assigns a graph structure to the protein set such a way that each protein has a

* Research supported by the Hungarian National Research Funds (OTKA), Project no. PD103921. The work of S.M. Szilágyi was supported by the TÁMOP-4.2.2.C-11/1/KONV-2012-0001 project, which is funded by the European Union, co-financed by the European Social Fund.

corresponding node. Edge weights are stored in the so-called similarity matrix S, which acts as a stochastic matrix. At any moment, edge weight s_{ij} reflects the posterior probability that protein i and protein j have a common evolutionary ancestor. TRIBE-MCL is an iterative algorithm, performing in each loop two main operations on the similarity matrix: inflation and expansion. Inflation raises each element of the similarity matrix to power $r > 1$, which is a previously established fixed inflation rate, favoring higher similarity values in the detriment of lower ones. Expansion, performed by raising matrix S to the second power, is aimed to favor longer walks along the graph. Further operations like column or row normalization, and matrix symmetrization are included to serve the stability and robustness of the algorithm, and to enforce the probabilistic constraint. Similarity values that fall below a previously defined threshold value ε are rounded to zero. Clusters are obtained as connected subgraphs in the graph.

Handling matrices of tens or hundreds of thousand rows and columns is prohibitively costly in both runtime and storage space. Recent fast TRIBE-MCL implementations (e.g. [10]) significantly reduced runtime, but the memory limitations still exist. The main goal of this paper is to introduce a novel fast TRIBE-MCL approach that uses only sparse matrices to store similarity values and a one-dimensional array to store intermediate values of a single row during expansion. This change can upgrade the size of processable data sets by an order of magnitude, and may also improve processing speed. The proposed method will be validated using the protein sequences of the SCOP95 database [7,4,2], and larger synthetic protein data sets [8] derived from SCOP95.

The remainder of this paper is structured as follows. Section 2 presents the details of the proposed efficient TRIBE-MCL algorithm. Section 3 thoroughly evaluates the behavior of the proposed method and discusses the achieved results and outlines the role of each parameter, while section 4 concludes this study.

2 Methods

In this paper we introduce a highly efficient implementation of the TRIBE-MCL algorithm, which also focuses on requiring reduced amount of memory storage. Any kind of TRIBE-MCL needs two instances of the similarity matrix: inflation, normalization, and symmetrization can be performed in a single matrix, but the expansion needs separate matrix instance for the input and the output data. Quick solutions existing so far use a sparse matrix and a two-dimensional array, which is not quite effective in reducing memory needs. The solution introduced here employs two instances of sparse matrix and an extra array that stores a single line of the similarity matrix during expansion.

The data structure employed for sparse matrix representation, shown in Fig. 1, is similar to the sparse supermatrix introduced in our previous work [10]. Each nonzero element of the sparse matrix (s_{ij}) is stored together with an approximated value of its symmetrically situated element in the matrix ($t_{ij} \approx s_{ji}$). Rows are stored concatenated in an array of records, each such record describing a nonzero element in the matrix. The starting address of each row is stored

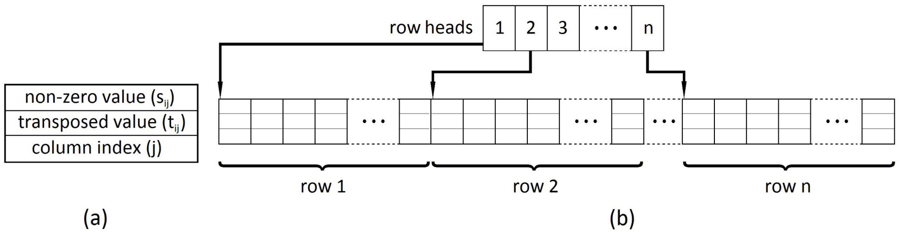

Fig. 1. The employed sparse matrix data structure: (a) the record for storing a single nonzero element; (b) array containing concatenated rows of the matrix

in a separate array of row heads. This whole data structure assures easy parsing of the matrix for all operations. With the exception of expansion, none of the operations increases the nonzero elements in any row. Normalization and inflation keeps the amount of nonzero elements constant, while symmetrization eliminates values under the chosen threshold ε. During the expansion, the sparse matrix is completely rewritten, as presented in Section 2.3.

2.1 Inflation

During inflation the sparse matrix is parsed record by record and both the s_{ij} similarity value and its approximated transposed value t_{ij} are raised to the r-th power. New values overwrite old ones in the same sparse matrix.

2.2 Normalization and Symmetrization

Normalization requires parsing the sparse matrix twice. In a first step, the sum of the similarity values is computed in each row. Let us denote by σ_i the sum of values in row i, $\forall i = 1 \dots n$, computed as: $\sigma_i = \sum_{j \in \text{row}_i} s_{ij}$. In the second step, each element in the sparse matrix is divided by the corresponding sum: $s_{ij}^{(\text{new})} = s_{ij}\sigma_i^{-1}$ and $t_{ij}^{(\text{new})} = t_{ij}\sigma_j^{-1}$, $\forall i = 1 \dots n$ and $\forall j \in \text{row}_i$. New similarity values overwrite the old ones in the same instance of sparse matrix.

The approximate symmetry of the similarity matrix S is assured by an iterative process in the main loop, situated between inflation and expansion. In each main loop, the symmetrization step is performed q times, and each symmetrization step is followed by a normalization. One symmetrization step requires a single parsing of the sparse matrix, and computes the following:

$$\begin{aligned} s_{ij}^{(\text{new})} &= \begin{cases} \sqrt{s_{ij}t_{ij}} & \text{if } s_{ij}t_{ij} \geq \varepsilon^2 \\ 0 & \text{otherwise} \end{cases} \qquad \begin{array}{l} \forall i = 1 \dots n, \\ \forall j \in \text{row}_i. \end{array} \\ t_{ij}^{(\text{new})} &= s_{ij}^{(\text{new})} \end{aligned} \tag{1}$$

Parameter q determines how accurate is the approximation of matrix symmetry, while ε is responsible for neglecting unimportant low values of similarity. Neglected values are not just overwritten by zero but completely eliminated from the sparse matrix, so that the time consuming expansion can operate on as few data as possible.

2.3 Expansion

Expansion is the only operation in the whole algorithm, which may raise the number of nonzeros in the similarity matrix, and thus requires a complete rewriting of the sparse matrix. Further on, our approach computes the expanded matrix row by row, and may need all rows of the input matrix as long as the last row of the output gets computed. This way we need two instances of the sparse matrix.

As long as a row of the expanded matrix is getting computed, it is stored in an n-element array. When the row is ready, nonzeros are transferred into the output sparse matrix. Let us denote the elements of this output matrix by $\overline{s}_{ij}$, $i, j = 1 \ldots n$, while $\mathbf{s}_i$ and $\overline{\mathbf{s}}_i$ will stand for row i of the input and output matrix, respectively. Since $\forall i, j = 1 \ldots n$, $\overline{s}_{ij} = \sum_{k \in \mathrm{row}_i} s_{ik} s_{kj}$, we may compute row with index i ($i = 1 \ldots n$) as

$$\overline{\mathbf{s}}_i = \left(\sum_{k \in \mathrm{row}_i} s_{ik} s_{k1} \sum_{k \in \mathrm{row}_i} s_{ik} s_{k2} \ldots \sum_{k \in \mathrm{row}_i} s_{ik} s_{kn} \right) = \sum_{k \in \mathrm{row}_i} s_{ik} \mathbf{s}_k \ . \tag{2}$$

Thus by parsing row i and computing a linear combination of rows with index k, where $k \in \mathrm{row}_i$ we obtain a whole row of the expansions' output matrix. Even after having parsed all rows and obtained $\overline{\mathbf{s}}_i$ ($\forall i = 1 \ldots n$), this is only half the job of expansion, because the new $\overline{t}_{ij}$ values are also needed in the next iteration. These $\overline{t}_{ij}$ values are obtained from the output sparse matrix. A pointer to the current element in each row is needed, initially set to the first element of the row. Then the sparse matrix is parsed row by row, and for each $\overline{s}_{ij}$ existing nonzero in the sparse matrix, the transposed value should be the current element pointed in row j. If there is a correspondence in coordinates, $\overline{t}_{ij}$ gets the pointed $\overline{s}_{ji}$ value, and the pointer in row j steps to the next element. When the row parsing gets to the end of the last row, each current element pointer reaches the end of its own row, and all nonzeros have received their transposed values.

2.4 Algorithm

Let us summarize the proposed approach of the TRIBE-MCL algorithm:

1. Initialize the parameters of the algorithm with the desired values: inflation rate r, similarity threshold ε, and symmetrization steps in each loop q.
2. Load initial similarity matrix and store it as a sparse matrix.
3. Normalize the values in the sparse matrix as described in section 2.2.
4. Perform inflation as described in section 2.1.
5. Normalize and symmetrize the sparse matrix in q steps as indicated in section 2.2, beginning and ending with normalization.
6. Perform expansion as presented in section 2.3.
7. Go back to step 4 unless convergence is achieved.
8. Clusters are obtained as isolated subgraphs in the final similarity graph.

At the convergence point, all isolated subgraphs are complete with approximately equal edge weights within the group.

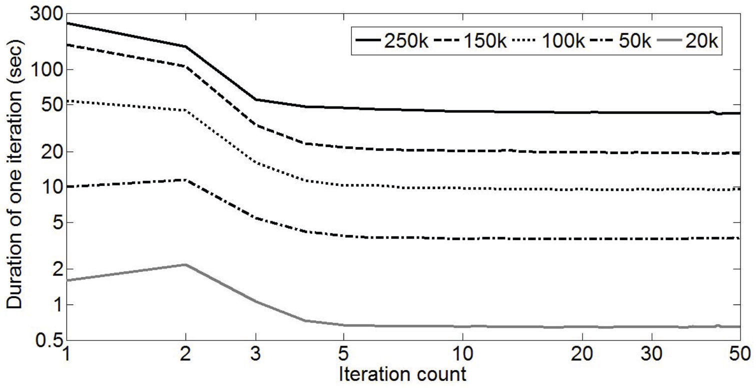

Fig. 2. Duration of first 50 iterations for various test data sizes at $r = 1.5$ and $\varepsilon = 10^{-3}$. The test with 250,000-item data set used $r = 1.7$.

3 Results and Discussion

The proposed algorithm was engaged in a series of numerical tests using synthetic protein data sets of various sizes in the range of 20-250 thousand items, created with the method given in [8]. For each test data size, 15 instances were created and the one with median number of nonzero values was chosen for the test.

Figure 2 exhibits the duration of each of the first fifty iterations in case of various matrix sizes, at inflation rate fixed at $r = 1.5$ and similarity threshold $\varepsilon = 10^{-3}$. As long as most nodes of the graph are connected together, namely in the first 5-6 loops, the computational load is somewhat higher, but it considerably falls thereafter, being virtually constant and low from the 10th loop.

Figure 3 exhibits the effect of the main parameters on the computational load of the algorithm. The input data here consisted of 50 thousand items having a similarity matrix of median density. Figure 3(a) indicates the total runtime of clustering performed in 50 iterations. As the inflation rate grows, the similarity matrix becomes sparser and thus the total runtime and also the length of late iterations is shorter. A lower value of the similarity threshold keeps small similarity values longer in the matrix, and consequently the processing needs more time (Fig. 3(b)).

The ratio between total runtime (duration of 50 iterations) and the length of a late iteration, exhibited in Fig. 3(c) shows us how much longer the first iterations are compared to late ones. This ratio would be 50 if the first loops were not at all computationally harder than the late ones. At $\varepsilon = 10^{-3}$ this ratio stays below 60, indicating that the algorithm quickly gets rid of unnecessary edges in the graph. At $\varepsilon = 10^{-4}$ this ratio can become 100, indicating that lots of computations are done in the first iterations to tear the graph into isolated subgraphs. Considering the fact that final clusters hardly differ from $\varepsilon = 10^{-4}$ to $\varepsilon = 10^{-3}$, choosing a low similarity threshold proves to be a waste of time.

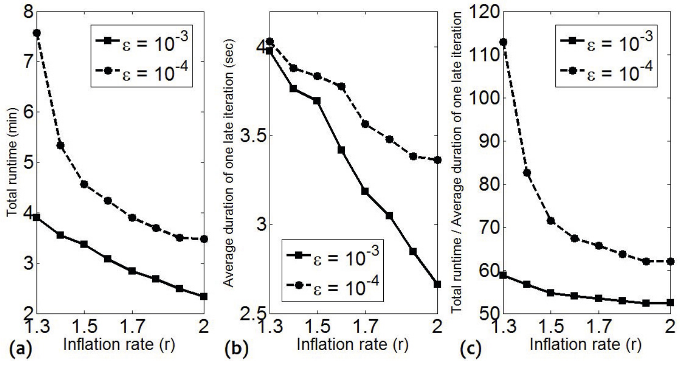

Fig. 3. Benchmark figures for a median density data set of 50k items, showing the influence of r and ε: (a) total runtime plotted vs. inflation rate; (b) average duration of one late iteration plotted vs. inflation rate; (c) the ratio between total runtime and duration of a late iteration plotted vs. inflation rate

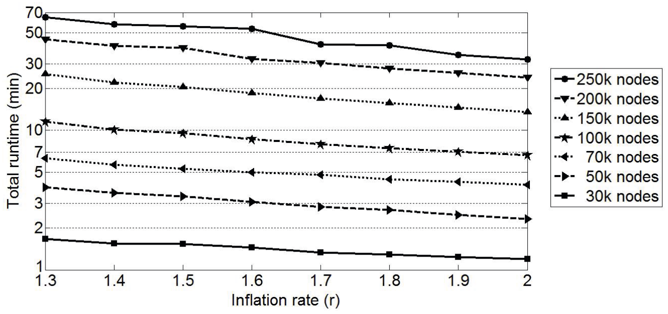

Fig. 4. Total runtime on various test data sizes, plotted against inflation rate, at constant similarity threshold value $\varepsilon = 10^{-3}$

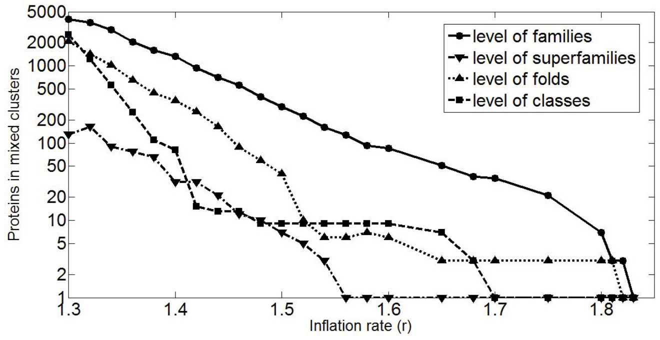

Fig. 5. The amount of proteins in mixed clusters found in a median density data set of 50k items, plotted against inflation rate, at constant $\varepsilon = 10^{-3}$

Figure 4 exhibits total runtime values plotted against inflation rate for input data set sizes ranging from 30,000 to 250,000 items, having the similarity threshold fixed at $\varepsilon = 10^{-3}$. Clustering at $r = 1.3$ takes 50-100% more time than at $r = 2$. Data sets of up to 250k items can be processed in an hour. Doubling the data size seemingly leads to four times longer processing.

Runtime benchmarks indicate that the proposed algorithm performs clustering in 30% less time than our previous high speed solution [10]. All efficiency tests were carried out on a notebook with quad core Haswell i7 processor running at 2.2GHz frequency and 24GB RAM memory, using a single core of the microprocessor.

The main goal of protein clustering is to reveal hidden similarities among proteins. When evaluating the output partition quality, one can count the number of mixed clusters (those which contain proteins from two or more different families) and their cardinality. We have shown in the previous work [9,10], that the inflation rate is the main factor to influence the amount of mixed clusters. The approach proposed here computes the same partitions as the conventional TRIBE-MCL, in a seriously more efficient way. Having drastically reduced the execution time of conventional TRIBE-MCL enabled us to perform a very detailed evaluation of the amount and nature of the mixed clusters in the output partition using synthetic data sets of several tens of thousand items.

Our synthetic data sets inherited the most important properties of SCOP95 proteins database, in which the proteins are organized in four-level hierarchy: classes, folds, superfamilies and families. Test data sets contain up to two dozens of classes, each of them containing several folds, the majority of which contain several superfamilies that are structured into families of proteins. Clusters established by TRIBE-MCL are called pure if all members belong to the same family. Alternately, there can be mixed clusters, which contain proteins of various families. Further on, mixed clusters may be mixtures of any of the four levels: e.g. if a mixed cluster contains proteins of different folds of a single class, then the mixture is called of the level of folds. The partition quality of a clustering algorithm may be characterized by the number of mixtures it produces, and their distribution among the four levels.

Figure 5 gives a detailed report on mixed clusters found in a 50,000-item data set of median density, for various inflation rates and a similarity threshold $\varepsilon = 10^{-3}$. The amount of mixed proteins highly depends on the inflation rate r, and in a very modest way on the similarity threshold ε. As the inflation rate rises, the number of mixed clusters and their cardinality drops quickly. At $r = 1.7$ there are no more mixtures at the level of classes and above $r = 1.83$ there are no more mixtures of any kind. As an effort to extract biologically useful information, it is recommendable to choose such an inflation rate, which produces mixtures and reveals hidden similarities among proteins of various known or unknown origins. Depending on the level and amount of mixtures we are looking for, the ideal inflation rate should be chosen in the range $1.4 - 1.7$.

The upper limit of processable data size is constrained by the memory of the used computer. The main determining factor is the number of nonzero values in the similarity matrix after the first expansion. After 2-3 iterations, the memory necessity stabilizes at a much lower level. With the current version of the algorithm, an ordinary PC with 2GB RAM can process a graph of 30,000 nodes, while 250,000 nodes require an upper class PC with 24GB RAM.

4 Conclusions

In this paper we have proposed an efficient approach to the graph-based TRIBE-MCL clustering method, a useful tool in protein sequence classification. The proposed approach proved extremely quick, and its memory needs are strongly reduced since previous versions. This novel implementation represents a major step of TRIBE-MCL towards handling huge data sets in reasonable time. We have shown that in case of the SCOP95 database and synthetic data sets derived from it, the most useful biological information can be extracted in case of inflation rates in range of $1.4 - 1.7$. Further enhancement of the algorithm's efficiency may be achieved via parallel implementation in CPUs or GPUs.

References

1. Altschul, S.F., Madden, T.L., Schaffen, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J.: Gapped BLAST and PSI-BLAST: A new generation of protein database search program. Nucl. Acids Res. 25, 3389–3402 (1997)
2. Andreeva, A., Howorth, D., Chadonia, J.M., Brenner, S.E., Hubbard, T.J.P., Chothia, C., Murzin, A.G.: Data growth and its impact on the SCOP database: New developments. Nucl. Acids Res. 36, D419–D425 (2008)
3. Enright, A.J., van Dongen, S., Ouzounis, C.A.: An efficient algorithm for large-scale detection of protein families. Nucl. Acids Res. 30, 1575–1584 (2002)
4. Lo Conte, L., Ailey, B., Hubbard, T.J., Brenner, S.E., Murzin, A.G., Chothia, C.: SCOP: A structural classification of protein database. Nucl. Acids Res. 28, 257–259 (2000)
5. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. 48, 443–453 (1970)
6. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. J. Mol. Biol. 147, 195–197 (1981)
7. Structural Classification of Proteins database, `http://scop.mrc-lmb.cam.ac.uk/scop`
8. Szilágyi, L., Kovács, L., Szilágyi, S.M.: Synthetic test data generation for hierarchical graph clustering methods. In: Loo, C.K., Yap, K.S., Wong, K.W., Teoh, A., Huang, K. (eds.) ICONIP 2014, Part II. LNCS, vol. 8835, pp. 303–310. Springer, Heidelberg (2014)
9. Szilágyi, L., Medvés, L., Szilágyi, S.M.: A modified Markov clustering approach to unsupervised classification of protein sequences. Neurocomput. 73, 2332–2345 (2010)
10. Szilágyi, S.M., Szilágyi, L.: A fast hierarchical clustering algorithm for large-scale protein sequence data sets. Comput. Biol. Med. 48, 94–101 (2014)